Cache Design

EJ Kim

Based on the lecture by Prof. David Culler and Prof. Patterson
Associativity vs Cycle Time

- Beware: Execution time is only final measure!
- Why is cycle time tied to hit time?

- Will Clock Cycle time increase?
  - Hill [1988] suggested hit time for 2-way vs. 1-way external cache +10%, internal + 2%
  - suggested big and dumb caches

Effective cycle time of assoc
pzbkski ISCA
Example: Avg. Memory Access Time vs. Miss Rate

- Example: assume CCT = 1.10 for 2-way, 1.12 for 4-way, 1.14 for 8-way vs. CCT direct mapped

<table>
<thead>
<tr>
<th>Cache Size (KB)</th>
<th>1-way</th>
<th>2-way</th>
<th>4-way</th>
<th>8-way</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.33</td>
<td>2.15</td>
<td>2.07</td>
<td>2.01</td>
</tr>
<tr>
<td>2</td>
<td>1.98</td>
<td>1.86</td>
<td>1.76</td>
<td>1.68</td>
</tr>
<tr>
<td>4</td>
<td>1.72</td>
<td>1.67</td>
<td>1.61</td>
<td>1.53</td>
</tr>
<tr>
<td>8</td>
<td>1.46</td>
<td>1.48</td>
<td>1.47</td>
<td>1.43</td>
</tr>
<tr>
<td>16</td>
<td>1.29</td>
<td>1.32</td>
<td>1.32</td>
<td>1.32</td>
</tr>
<tr>
<td>32</td>
<td>1.20</td>
<td>1.24</td>
<td>1.25</td>
<td>1.27</td>
</tr>
<tr>
<td>64</td>
<td>1.14</td>
<td>1.20</td>
<td>1.21</td>
<td>1.23</td>
</tr>
<tr>
<td>128</td>
<td>1.10</td>
<td>1.17</td>
<td>1.18</td>
<td>1.20</td>
</tr>
</tbody>
</table>

(Red means A.M.A.T. not improved by more associativity)
Fast Hit Time + Low Conflict => Victim Cache

- How to combine fast hit time of direct mapped yet still avoid conflict misses?
- Add buffer to place data discarded from cache
- Jouppi [1990]: 4-entry victim cache removed 20% to 95% of conflicts for a 4 KB direct mapped data cache
- Used in Alpha, HP machines
Reducing Misses via “Pseudo-Associativity”

- How to combine fast hit time of Direct Mapped and have the lower conflict misses of 2-way SA cache?
- Divide cache: on a miss, check other half of cache to see if there, if so have a pseudo-hit (slow hit)

**Drawback:** CPU pipeline is hard if hit takes 1 or 2 cycles
- Better for caches not tied directly to processor (L2)
- Used in MIPS R1000 L2 cache, similar in UltraSPARC
Reducing Misses by **Hardware**
Prefetching of Instructions & Data

- **E.g., Instruction Prefetching**
  - Alpha 21064 fetches 2 blocks on a miss
  - Extra block placed in “stream buffer”
  - On miss check stream buffer

- **Works with data blocks too:**
  - Jouppi [1990] 1 data stream buffer got 25% misses from 4KB cache; 4 streams got 43%
  - Palacharla & Kessler [1994] for scientific programs for 8 streams got 50% to 70% of misses from 2 64KB, 4-way set associative caches

- Prefetching relies on having extra memory bandwidth that can be used without penalty
Reducing Misses by Software Prefetching Data

• Data Prefetch
  - Load data into register (HP PA-RISC loads)
  - Cache Prefetch: load into cache (MIPS IV, PowerPC, SPARC v. 9)
  - Special prefetching instructions cannot cause faults; a form of speculative execution

• Prefetching comes in two flavors:
  - Binding prefetch: Requests load directly into register.
    » Must be correct address and register!
  - Non-Binding prefetch: Load into cache.
    » Can be incorrect. Faults?

• Issuing Prefetch Instructions takes time
  - Is cost of prefetch issues < savings in reduced misses?
  - Higher superscalar reduces difficulty of issue bandwidth
Reducing Misses by Compiler Optimizations

- McFarling [1989] reduced caches misses by 75% on 8KB direct mapped cache, 4 byte blocks in software.
- Instructions
  - Reorder procedures in memory so as to reduce conflict misses
  - Profiling to look at conflicts (using tools they developed)
- Data
  - Merging Arrays: improve spatial locality by single array of compound elements vs. 2 arrays
  - Loop Interchange: change nesting of loops to access data in order stored in memory
  - Loop Fusion: Combine 2 independent loops that have same looping and some variables overlap
  - Blocking: Improve temporal locality by accessing “blocks” of data repeatedly vs. going down whole columns or rows
Merging Arrays Example

/* Before: 2 sequential arrays */
int val[SIZE];
int key[SIZE];

/* After: 1 array of structures */
struct merge {
    int val;
    int key;
};
struct merge merged_array[SIZE];

Reducing conflicts between val & key;
improve spatial locality
Loop Interchange Example

/* Before */
for (k = 0; k < 100; k = k+1)
  for (j = 0; j < 100; j = j+1)
    for (i = 0; i < 5000; i = i+1)
      x[i][j] = 2 * x[i][j];
/* After */
for (k = 0; k < 100; k = k+1)
  for (i = 0; i < 5000; i = i+1)
    for (j = 0; j < 100; j = j+1)
      x[i][j] = 2 * x[i][j];

Sequential accesses instead of striding through memory every 100 words; improved spatial locality
/* Before */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
    a[i][j] = 1/b[i][j] * c[i][j];
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
    d[i][j] = a[i][j] + c[i][j];

/* After */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
  {
    a[i][j] = 1/b[i][j] * c[i][j];
    d[i][j] = a[i][j] + c[i][j];
  }

2 misses per access to a & c vs. one miss per access; improve spatial locality
Blocking Example

/* Before */
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
        {r = 0;
         for (k = 0; k < N; k = k+1){
             r = r + y[i][k]*z[k][j];
         }
         x[i][j] = r;
    }

• Two Inner Loops:
  - Read all NxN elements of z[]
  - Read N elements of 1 row of y[] repeatedly
  - Write N elements of 1 row of x[]

• Capacity Misses a function of N & Cache Size:
  - $2N^3 + N^2$ => (assuming no conflict; otherwise ...)

• Idea: compute on BxB submatrix that fits
Blocking Example

/* After */
for (jj = 0; jj < N; jj = jj+B)
for (kk = 0; kk < N; kk = kk+B)
for (i = 0; i < N; i = i+1)
  for (j = jj; j < min(jj+B-1,N); j = j+1)
    {r = 0;
     for (k = kk; k < min(kk+B-1,N); k = k+1) {
       r = r + y[i][k]*z[k][j];
     }
     x[i][j] = x[i][j] + r;
    }

• B called Blocking Factor
• Capacity Misses from $2N^3 + N^2$ to $N^3/B+2N^2$
• Conflict Misses Too?
Summary of Compiler Optimizations to Reduce Cache Misses (by hand)

- vpenta (nasa7)
- gmtty (nasa7)
- tomcatv
- btrix (nasa7)
- mxm (nasa7)
- spice
- cholesky
- (nasa7)
- compress

Performance Improvement:
- merged arrays
- loop interchange
- loop fusion
- blocking
Summary: Miss Rate Reduction

\[ \text{CPU time} = IC \times \left( \text{CPI}_{\text{Execution}} + \frac{\text{Memory accesses}}{\text{Instruction}} \right) \times \text{Miss rate} \times \text{Miss penalty} \times \text{Clock cycle time} \]

- 3 Cs: Compulsory, Capacity, Conflict
  0. Larger cache
  1. Reduce Misses via Larger Block Size
  2. Reduce Misses via Higher Associativity
  3. Reducing Misses via Victim Cache
  4. Reducing Misses via Pseudo-Associativity
  5. Reducing Misses by HW Prefetching Instr, Data
  6. Reducing Misses by SW Prefetching Data
  7. Reducing Misses by Compiler Optimizations

- Prefetching comes in two flavors:
  - Binding prefetch: Requests load directly into register.
    - Must be correct address and register!
  - Non-Binding prefetch: Load into cache.
    - Can be incorrect. Frees HW/SW to guess!
Review: Improving Cache Performance

1. Reduce the miss rate,
2. Reduce the miss penalty, or
3. Reduce the time to hit in the cache.
Write Policy: Write-Through vs Write-Back

- **Write-through**: all writes update cache and underlying memory/cache
  - Can always discard cached data - most up-to-date data is in memory
  - Cache control bit: only a *valid* bit
- **Write-back**: all writes simply update cache
  - Can’t just discard cached data - may have to write it back to memory
  - Cache control bits: both *valid* and *dirty* bits

**Other Advantages:**
- **Write-through**:
  » memory (or other processors) always have latest data
  » Simpler management of cache
- **Write-back**:
  » much lower bandwidth, since data often overwritten multiple times
  » Better tolerance to long-latency memory?
Write Policy 2: Write Allocate vs Non-Allocate (What happens on write-miss)

• Write allocate: allocate new cache line in cache
  - Usually means that you have to do a “read miss” to fill in rest of the cache-line!
  - Alternative: per/word valid bits

• Write non-allocate (or “write-around”):
  - Simply send write data through to underlying memory/cache - don't allocate new cache line!
1. Reducing Miss Penalty: Read Priority over Write on Miss
1. Reducing Miss Penalty: Read Priority over Write on Miss

- **Write-through w/ write buffers** => RAW conflicts with main memory reads on cache misses
  - If simply wait for write buffer to empty, might increase read miss penalty (old MIPS 1000 by 50%)
  - Check write buffer contents before read; if no conflicts, let the memory access continue

- **Write-back want buffer to hold displaced blocks**
  - Read miss replacing dirty block
  - Normal: Write dirty block to memory, and then do the read
  - Instead copy the dirty block to a write buffer, then do the read, and then do the write
  - CPU stall less since restarts as soon as do read
2. Reduce Miss Penalty: Early Restart and Critical Word First

- Don't wait for full block to be loaded before restarting CPU
  
  - **Early restart**—As soon as the requested word of the block arrives, send it to the CPU and let the CPU continue execution
  
  - **Critical Word First**—Request the missed word first from memory and send it to the CPU as soon as it arrives; let the CPU continue execution while filling the rest of the words in the block. Also called *wrapped fetch* and *requested word first*

- Generally useful only in large blocks,

- Spatial locality => tend to want next sequential word, so not clear if benefit by early restart
3. Reduce Miss Penalty: Non-blocking Caches to reduce stalls on misses

- **Non-blocking cache** or **lockup-free cache** allow data cache to continue to supply cache hits during a miss
  - requires F/E bits on registers or out-of-order execution
  - requires multi-bank memories
- "hit under miss" reduces the effective miss penalty by working during miss vs. ignoring CPU requests
- "hit under multiple miss" or "miss under miss" may further lower the effective miss penalty by overlapping multiple misses
  - Significantly increases the complexity of the cache controller as there can be multiple outstanding memory accesses
  - Requires multiple memory banks (otherwise cannot support)
  - Pentium Pro allows 4 outstanding memory misses
4: Add a second-level cache

• L2 Equations
  \[ \text{AMAT} = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times \text{Miss Penalty}_{L1} \]
  \[ \text{Miss Penalty}_{L1} = \text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2} \]
  \[ \text{AMAT} = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times (\text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2}) \]

• Definitions:
  - **Local miss rate**—misses in this cache divided by the total number of memory accesses to this cache (Miss rate\(_{L2}\))
  - **Global miss rate**—misses in this cache divided by the total number of memory accesses generated by the CPU
  - Global Miss Rate is what matters
Comparing Local and Global Miss Rates

- 32 KByte 1st level cache; Increasing 2nd level cache
- Global miss rate close to single level cache rate provided L2 >> L1
- Don't use local miss rate
- L2 not tied to CPU clock cycle!
- Cost & A.M.A.T.
- Generally Fast Hit Times and fewer misses
- Since hits are few, target miss reduction
Reducing Misses:
Which apply to L2 Cache?

• Reducing Miss Rate
  1. Reduce Misses via Larger Block Size
  2. Reduce Conflict Misses via Higher Associativity
  3. Reducing Conflict Misses via Victim Cache
  4. Reducing Conflict Misses via Pseudo-Associativity
  5. Reducing Misses by HW Prefetching Instr, Data
  6. Reducing Misses by SW Prefetching Data
  7. Reducing Capacity/Conf. Misses by Compiler Optimizations
L2 cache block size & A.M.A.T.
Relative CPU Time

- 32KB L1, 8 byte path to memory
Reducing Miss Penalty Summary

\[ \text{CPUtime} = IC \times \left( CPI_{\text{Execution}} + \frac{\text{Memory accesses}}{\text{Instruction}} \right) \times \text{Miss rate} \times \text{Miss penalty} \times \text{Clock cycle time} \]

- **Four techniques**
  - Read priority over write on miss
  - Early Restart and Critical Word First on miss
  - Non-blocking Caches (Hit under Miss, Miss under Miss)
  - Second Level Cache

- **Can be applied recursively to Multilevel Caches**
  - Danger is that time to DRAM will grow with multiple levels in between
  - First attempts at L2 caches can make things worse, since increased worst case is worse
What is the Impact of What You’ve Learned About Caches?

• 1960-1985: Speed = \( f(\text{no. operations}) \)
• 1990
  - Pipelined Execution & Fast Clock Rate
  - Out-of-Order execution
  - Superscalar Instruction Issue
• 1998: Speed = \( f(\text{non-cached memory accesses}) \)
• Superscalar, Out-of-Order machines hide L1 data cache miss (-5 clocks) but not L2 cache miss (-50 clocks)?
## Cache Optimization Summary

<table>
<thead>
<tr>
<th>Technique</th>
<th>MR</th>
<th>MP</th>
<th>HT</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Larger Block Size</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>Higher Associativity</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>Victim Caches</td>
<td>+</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Pseudo-Associative Caches</td>
<td>+</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>HW Prefetching of Instr/Data</td>
<td>+</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Compiler Controlled Prefetching</td>
<td>+</td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Compiler Reduce Misses</td>
<td>+</td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Priority to Read Misses</td>
<td>+</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Early Restart &amp; Critical Word 1st</td>
<td>+</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Non-Blocking Caches</td>
<td>+</td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Second Level Caches</td>
<td>+</td>
<td></td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>